논문 01-26-12C-7 한국통신학회논문지 '01-12 Vol.26 No.12C

고속 네트웍 기반의 분산병렬시스템에서의 성능 향상 분석 모델

정회원 김 화 성*

Speedup Analysis Model for High Speed Network based Distributed Parallel Systems

Hwa-sung Kim* *Regular Members*

요 약

분산병렬처리의 목적은 다양한 내재 병렬 형태의 특징을 갖는 연산 집약적 문제를 고속 네트웍으로 연결되어진 다수의 고성능 및 병렬 컴퓨터들의 각기 다른 능력을 최대한 이용하여 해결함에 있다. 본 논문에서는 분산병렬시스템을 이용하는 경우의 성능 향상 분석을 위해 일반적인 그래프 표현 방법을 포함하는 계산 모델을 제안하고 프로그램의 수행을 위한 스케쥴링 시에 성능 향상이 어떠한 요인에 의해 달성되는지를 분석한다. 제안된 표현 방법은 동기종 및 이기종 시스템 모두에 적용되어질 수 있다. 분산병렬 시스템에서 스케쥴링을 통하여 더 많은 속도향상을 얻기 위해서는 태스크와 병렬 컴퓨터간의 병렬특성의 일치가 주의 깊게 다루어져야 하며 태스크의 이동으로 인한 통신 오버헤드가 최소화 되어야 한다.

ABSTRACT

The objective of Distributed Parallel Computing is to solve the computationally intensive problems, which have several types of parallelism, on a suite of high performance and parallel machines in a manner that best utilizes the capabilities of each machine. In this paper, we propose a computational model including the generalized graph representation method of distributed parallel systems for speedup analysis, and analyze how the super-linear speedup is achieved when scheduling of programs with diverse embedded parallelism modes onto a distributed heterogeneous supercomputing network environment. The proposed representation method can also be applied to simple homogeneous or heterogeneous systems whose components are heterogeneous only in terms of the processor speed. In order to obtain the more speedup, the matching of the parallelism characteristics between tasks and parallel machines should be carefully handled while minimizing the communication overhead.

I. Introduction

The heterogeneous systems are composed of multiple dissimilar machines which corporate in solving a problem. This paper focuses on a special type of the heterogeneous systems named as Distributed Parallel Systems which consist of a set of loosely coupled parallel and vector machines connected via high speed networks. The objective of distributed parallel processing is to solve computationally intensive problems that have several types of parallelism, on a suite of high performance and parallel machines in a manner that best utilizes the capabilities of each machine [1,2,3,4]. Recent advances in research on network protocols and transmissions, and innovations in

* 광운대학교 전자공학부 네트워크 컴퓨팅 연구실(hwkim@daisy.kwangwoon.ac.kr),
논문번호 : K01199-0914, 접수일자 : 2001년 9월 14일
※ 본 연구는 광운대학교 2000년도 교내 학술연구비 지원에 의해 수행되었습니다.

supercomputer design have made practical the development of high-performance applications whose processing is distributed over several supercomputers. These applications make use of the combined computational power of several resources to increase their performance, and exploit the heterogeneity of diverse architectures and software systems by assigning selected tasks to platforms which can best support them.

Actually, many large-scale scientific applications have more than one type of embedded parallelism, such as SIMD, MIMD, and vector parallelism types, in its various code segments [5,6]. Since it is unlikely that a single parallel machine would execute the mixed-type of computations with the maximum possible speedup, a homogeneous system can not achieve the optimal speedup for these applications. Therefore, it is more efficient to allocate the different segments of a program with different types of parallelism to various parallel machines that can execute the assigned segments with the optimal speedup. In this case, however, the network overhead required for using the various machines should not offset the advantage obtained by assigning the segments of the program to the machines with matching parallelism type.

Unlike homogeneous systems, a super-linear speedup (more than a linear speedup) can be achieved when using a distributed parallel system. Donaldson et al. [4] defined the notions of super-linear speedup and showed that the speedup consists of heterogeneous and parallel components in general terms without mentioning how the heterogeneous component can be analyzed in detail. The rest of this paper is organized as follows: Section 2 formulates the scheduling problem in distributed parallel systems. Section 3 analyzes the speedup when scheduling in distributed paralleled systems. Finally, Section 4 presents the conclusion.

Ⅱ. Formulation of the Problem

The distributed parallel system used in this paper is a point-to-point network of multiple high performance and parallel machines, each of which has a specific parallelism type. The distributed parallel system is represented by a *Network Graph* as defined below.

Definition 1: A distributed parallel system H is represented by an undirected network graph $G_H = <V_H, E_H>$, where $V_H = \{M_{i,k}, i \in \{1, 2, \ldots, m\}, k \in \{1=SISD, 2=SIMD, 3=MIMD, 4=vecto r, \ldots\}\}$. For the convenience, the machine is sometimes denoted by M_i (dropping the integer k) when the parallelism type of the machine is not important in the discussion. Each machine $M_{i,k}$ in V_H is associated with a tuple - $[k, R_{i,k}]$ where k is the parallelism type of machine M_i and $R_{i,k}$ is the relative performance of machine $M_{i,k}$ to the fastest machine of type k among those in a distributed parallel system (if machine $M_{i,k}$ is the fastest among the machines of parallelism type k, then $R_{i,k} = 1$ otherwise $R_{i,k} > 1$). E_H is a set of

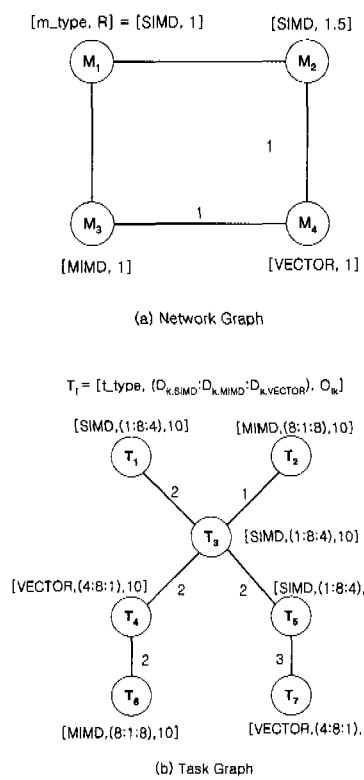


Fig. 1 Example of Network and Task Graph

edges; $E_H \subseteq V_H \times V_H$ such that $(M_i, M_j) \in E_H$ if M_i is connected to M_j through a direct link. Each edge is associated with an integer representing the communication cost that is incurred when sending a unit of information between the two machines, i.e. this integer is an estimate of the transfer rate and type conversion overhead between the machines neglecting the setup time. It is assumed that each machine in the heterogeneous network may execute at most one task at a time.

On the other hand, a parallel program consists of a number of cooperating and communicating tasks, each of which is characterized by a parallelism type such as SISD, SIMD, MIMD, or a vector type. The behavior of the program is described by a *Task Graph* as defined below.

Definition 2: A program P is represented by a directed, a cyclic task graph $G_P = <V_P, E_P>$ where $V_P = \{t_{i,k}, i \in \{1, 2, . . . , n\}, k \in \{1=SISD, 2=SIMD, 3=MIMD, 4=vector, . . .\}\}$ is a partition of P consisting of n tasks each of which has a parallelism type k. For the convenience, the task is sometimes only denoted by t_i (dropping the integer k). Each task t_{ik} in V_P is associated with a tuple - $[k, (D_{k,b}... D_{k,b}, ..., D_{k,l}), O_{i,k}]$ where $D_{k,l}$ is a coefficient that represents the type mismatch penalty between task $t_{i,k}$ and machine of type l (if $k = l$, then $D_{k,l} = 1$ otherwise $D_{k,l} > 1$), L is the number of available machine types in the distributed parallel system under consideration. $O_{i,k}$ is the optimal execution time of $t_{i,k}$ on the its best matching machine. E_P is a set of edges; $E_P \subseteq V_P \times V_P$ such that $(t_i, t_j) \in E_P$ if and only if t_i must execute before t_i due to data dependency and/or communications between the two tasks. Each edge is associated with an integer representing the amount of information to be transferred between two tasks. It is assumed that the execution of tasks in the machines is non-preemtive, i.e. once the task begins execution, it executes until its completion.

After mapping the tasks of a program P to the machines in a distributed parallel system H, another task graph called an *Allocated Task Graph* is obtained. In an allocated task graph, each task is assigned a single value, which is its execution time on a specific machine, and each edge is assigned the actual communication cost between two tasks. Let $T(t_{i,k}, M_{j,l})$ be the execution time of task $t_{i,k}$ on machine $M_{j,l}$. We can describe $T(t_{i,k}, M_{j,l})$ by the product: $D_{k,l} \times R_{j,l} \times O_{i,k}$. If the parallelism types of a task does not correspond to that of a machine, the execution time will be $D_{k,l}$ times the optimal execution time $O_{i,k}$. On the other hand, a distributed parallel system can be composed of machines with various types and various performance even in same type. Therefore, a task can be assigned to the machine, which is not the optimal choice even though their types match. $R_{j,l}$ incorporates the non-optimal machine choice although type matching succeeds. R_{jl} should be 1 if a parallel machine M_{jl} on which task t_{ik} is allocated is the fastest machine among the machines with parallelism type l. $O_{i,k}$ represents the optimal mapping of task t_i of parallelism type k to the best machine of type k.

Let $C(t_i, M_{j,b}, t_{i'}, M_{j',l'})$ be the actual communication cost between a task $t_{i,k}$ allocated to a machine $M_{j,l}$ and any its parent task $t_{i',k'}$ allocated to a machine $M_{j',l'}$. The parallel execution of tasks using various machines accompanies the communication overhead. Moreover in the distributed parallel systems, more speedup can be achieved by executing the task nodes which lie on a sequential path of a task graph on different machines. This will introduce the communication overhead. The communication overhead is decided by the amount of the data $A_{i,i'}$ transferred between any two tasks t_i and $t_{i'}$, the transmission rate $H_{j,j'}$ between any two machines M_j and $M_{j'}$, and the conversion time of the data representation and the synchronization time $F_{ll'}$ between any two machines of type l and l'. After all the tasks that are mapped to the same machine are totally ordered, the allocated task graph is changed into a *Scheduled Task Graph* in

which the mapping of tasks to machines in the network and any execution order between tasks are fixed. In a scheduled task graph, additional edges to represent the execution order between tasks assigned to the same machine can be added and the transitive edges are removed.

The scheduling problem can be described as finding a mapping function $\Pi$of the tasks of a program P into the machines of a distributed parallel system H, i.e. $\Pi: V_P \rightarrow V_H$, and the execution order of all the tasks that are mapped to the same machine such that the schedule length is minimized. The schedule length is equal to the summation of execution times $T(t_{i,k}, M_{j,l})$ and the communication costs $C(t_i, M_{j,l}, t_{i'}, M_{j',l'})$ along the critical path in a scheduled task graph. The critical path in a scheduled task graph is the longest path between any starting task and any ending task in the graph. In this case, the communication time between two tasks executing on the same machine is assumed to be zero time unit. Especially in the distributed parallel systems, the parallelism type matching heavily affects the schedule length.

Ⅲ. Speedup in Distributed Parallel Systems

The speedup that can be achieved when executing a parallel program P in a distributed parallel system H can be described by S_P such as

$$S_P = \frac{\min\{T_P^{M_1}, T_P^{M_2}, .., T_P^{M_i}, ..., T_P^{M_m}\}}{T_P^H} \quad (1)$$

where m is the number of distinct machines in the network, $T_P^{M_i}$ is the execution time when the whole program P is executed on single machine M_i, and T_P^H is the execution time of P in the distributed parallel system H. (i.e. the schedule). If $S_P = n$, a linear speedup is obtained as a result of running the program P in the distributed parallel system. On the other hand, if $S_P < n$, a sublinear speedup is obtained, while if $S_P > n$, a super-linear speedup is achieved. In the following, $min\{T_P^{M_1}, T_P^{M_2}, ..., T_P^{M_m}\}$ is abbreviated as $T_P^{M_F}$ for the convenience, and described as follows:

$$T_P^{M_F} = \min\{T_P^{M_1}, T_P^{M_2}, ..., T_P^{M_m}\} = \min_{1 \le j \le m} \sum_{1 \le i \le n} D_{k,l} \times R_{j,l} \times O_{i,k} \quad (2)$$

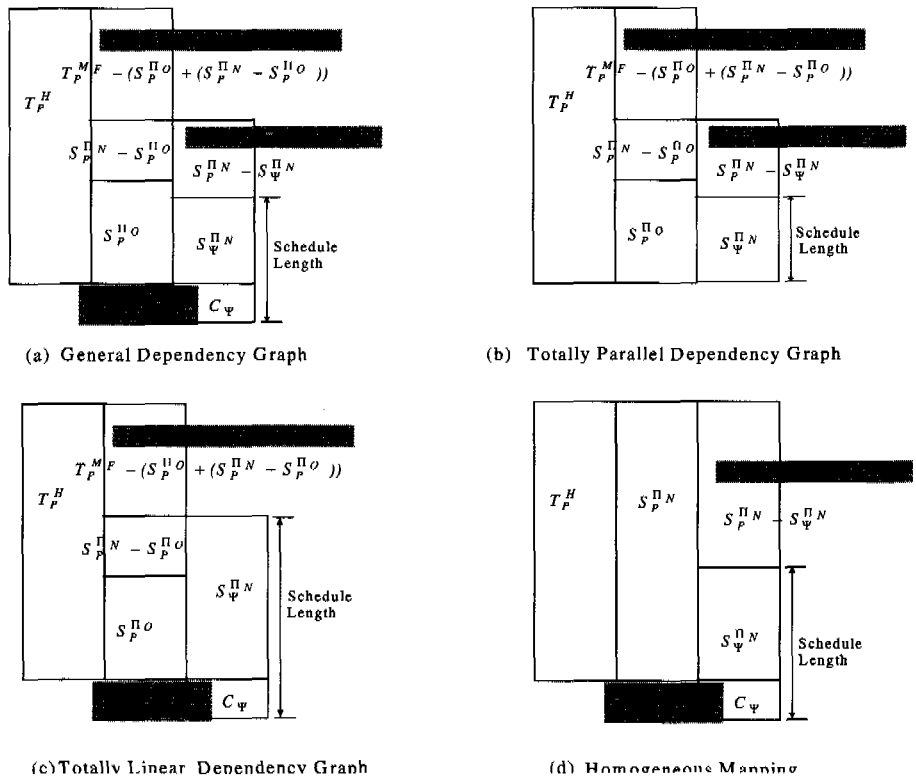
In Eq. (1), we can see that more speedup is expected as $T_P^{M_F}$ gets bigger or T_P^H gets smaller. $T_P^{M_F}$, as can be seen in Eq. (2), gets bigger according that the parallelism types of tasks and the machines in the distributed parallel system are more diverse such that the execution of tasks on the ill-matched machines incur more type mismatching penalties. Since all the tasks of a program P should be executed on single machine, in this case, the parallelism types of the most tasks of P will not be matched with that of the machine. And these type mismatching tasks need more execution time than the case in which those task are run on the machine whose parallelism types match with that of those tasks.

On the other hand, T_P^H is defined as the sum of the execution time $S_\Psi^{\Pi_N}$ of the tasks and sum of the communication cost C_Ψ of the edges along the critical path Ψ of the scheduled task graph derived from the general dependency task graph as follows.

$$T_P^H = S_\Psi^{\Pi_N} + C_\Psi \quad (3)$$

The definition of $S_\Psi^{\Pi_N}$ and C_Ψ is given in the following where Π represents the mapping function and Ψ is the critical path of the scheduled task graph. In this case, the mapping of the tasks need not to be optimal necessarily.

$$S_\Psi^{\Pi_N} = \sum_{\substack{t_i \in \Psi \\ M_j = \Pi_N(t_i)}} T(t_{i,k}, M_{j,l}) = \sum_{\substack{t_i \in \Psi \\ M_j = \Pi_N(t_i)}} D_{k,l} \times R_{j,l} \times O_{i,k} \quad (4)$$



(a) General Dependency Graph (b) Totally Parallel Dependency Graph

(c) Totally Linear Dependency Graph (d) Homogeneous Mapping

Fig. 2 Schematic Representation of Components in Speedup

$$C_\Psi = \sum_{\substack{t_i,t_{i'}\in\Psi \\ M_j=\Pi_N(t_i) \\ M_{j'}=\Pi_N(t_{i'})}} C(t_i,M_{j,l},t_{i'},M_{j',l'}) = \sum_{\substack{t_i,t_{i'}\in\Psi \\ M_j=\Pi_N(t_i) \\ M_{j'}=\Pi_N(t_{i'})}} H_{j,j'} \times F_{l,l'} \times A_{i,i'} \quad (5)$$

On the other hand, when some of the tasks of program P are not mapped optimally, sum of the execution time of all the tasks of program P after mapping is done is represented by $S_P^{\Pi_N}$. Also, the sum of the execution time of all the tasks of program P when their mapping is totally optimal, is represented by $S_P^{\Pi_O}$. The definition of $S_P^{\Pi_N}$ and $S_P^{\Pi_O}$ is given in the following.

$$S_P^{\Pi_N} = \sum_{\substack{1\le i\le n \\ M_j=\Pi_N(t_i)}} T(t_{i,k},M_{j,l}) = \sum_{\substack{1\le i\le n \\ M_j=\Pi_N(t_i)}} D_{k,l} \times R_{j,l} \times O_{i,k} \quad (6)$$

$$S_P^{\Pi_O} = \sum_{\substack{1\le i\le n \\ M_j=\Pi_O(t_i)}} T(t_{i,k},M_{j,l}) = \sum_{\substack{1\le i\le n \\ M_j=\Pi_O(t_i)}} O_{i,k} \quad (7)$$

Using the notation defined in Eq. (3)-(7), T_P^H can be rewritten as follows:

$$T_P^H = T_P^{M_F} - (T_P^{M_F} - (S_P^{\Pi_O} + (S_P^{\Pi_N} - S_P^{\Pi_O}))) - (S_P^{\Pi_N} - S_\Psi^{\Pi_N} - C_\Psi) \quad (8)$$

In Eq. (8), $S_P^{\Pi_N} - S_P^{\Pi_O}$ represents the penalty caused by non-optimal mapping, which depends on the matching degree of the parallelism types between the program tasks and the machines in the system and the relative performance of the machines as shown in Eq. (6), (7). This gets smaller according that more tasks are allocated to their type matching machines(if all the tasks are mapped optimally, it reduces to zero). On the other hand, $(S_P^{\Pi_O} + (S_P^{\Pi_N} - S_P^{\Pi_O}))$ is simply reduces to $S_P^{\Pi_N}$. $T_P^{M_F} - (S_P^{\Pi_O} + (S_P^{\Pi_N} - S_P^{\Pi_O}))$, which is the

difference between $T_P^{M_F}$ and $S_P^{\Pi_N}$, represents the gain caused by the heterogeneous mapping.

Once all the tasks are mapped to the machines, some of them except those allocated to the same machine can be executed in parallel according to the degree of parallelism inherent in program. $S_P^{\Pi_N} - S_\Psi^{\Pi_N} - C_\Psi$ represents such gain caused by parallel execution. C_Ψ is the amount of communication overhead as in Eq. (5).

In summary, the speedup increases according to three components as shown in Fig. 1: (a) more gain caused by *heterogeneous mapping*, (b) more gain by *parallel execution*; and (c) less overhead by communication. And the gain caused by heterogeneous mapping depends on three components: degree of parallelism type mismatching penalties between the program tasks and the machines in the system, the optimality of mapping which yields more parallelism type matching, and the relative performance of the machines.

If the tasks are totally parallel and the number of the tasks is not larger than that of the machines, the communication will not affect the speedup. T_P^H is same as the execution time of the task requiring the biggest time and is calculated as follows where n is the total number of the tasks:

$$T_P^H = \max_{\substack{1\le i\le N \\ M_j=\Pi_N(t_i)}} T(t_{i,k}, M_{j,l})$$

$$= T_P^{M_F} - (T_P^{M_F} - (S_P^{\Pi_O} + (S_P^{\Pi_N} - S_P^{\Pi_O}))) = -(S_P^{\Pi_N} - S_\Psi^{\Pi_N}) \quad (9)$$

Compared to the totally parallel case, the amount of speedup for totally linear tasks does not depend on the degree of the parallelism because it does not have the program modules that can be executed concurrently. T_P^H can be written as:

$$\sum_{\substack{1\le i\le n \\ M_j=\Pi(t_i)}} [T(t_{i,k}, M_{j,l}) + C(t_i, M_{j,l}, t_{i'}, M_{j',l'})]$$

$$= T_P^{M_F} - (T_P^{M_F} - (S_P^{\Pi_O} + (S_P^{\Pi_N} - S_P^{\Pi_O}))) + C_\Psi \quad (10)$$

In the homogeneous case, there is no gain obtained by the heterogeneous mapping because the machines in a network do not show any differences in terms of the parallelism type and the machine performance. Therefore, the $T_P^{M_F} - (S_P^{\Pi_O} + (S_P^{\Pi_N} - S_P^{\Pi_O}))$ term should be zero. The speedup in homogeneous case is only achieve d by the gain in parallel execution at the expense of the communication overhead.

Ⅳ. Conclusion

In this paper, we have presented a computation al model, which is the generalized graph represent ation method of distributed parallel systems for speedup analysis. The proposed representation met hod includes the task graph and the network graph. We also formulated the scheduling problem in the distributed parallel systems using the prop osed computational model. The various component s of speedup were also analyzed when scheduling a parallel program onto a distributed parallel system. The super-linear speedup can not be obtained by simply increasing the number of the similar machines because adding additional machines also increases the inter-machine communication and additional machines can not perform the optimal execution for the ill-matched type code either. In distributed parallel system, the super-linear speedup can be achieved by assigning segments of a program with different parallelism types to their best type matching available machines. The speedup is improved by larger degree of the parallelism in the program and by larger average type mismatching penalty in the network. On the contrary, the speedup is degraded by larger communication overhead. The communication overhead required for using a number of heterogeneous machines connected via a network should not offset the advantage obtained by assigning the segments of a program to the machines with matching type of

parallelism, otherwise the super-linear speedup can not be achieved.

References

[1] R.F. Freund and D.S. Conwell, Super-concurrency: "A form of distributed heterogeneous supercomputing" *Supercomputing Review*, Vol. 3, No.10, Oct. 1990, pp. 47-50

[2] B. Narahari, A. Youssef and H.A Choi, "Matching and Scheduling in a Generalized Optimal Selection Theory" Proc. *Workshop on Heterogeneous Processing*, April 1994, pp.3-8

[3] Rajkumar Buyya, High performance Cluster Computing: Architecture and Systems, Prentice Hall 1999

[4] V. Donaldson, F. Berman and R. Paturi, "Program Speedup in a Heterogeneous Computing" *Journal of Parallel and Distributed Computing*, Vol.21, 1994, pp. 316-322

[5] C.R. Mechoso, John D. Farrara and J.A. Spahr, "Achieving Superlinear Speedup on a Heterogeneous Distributed Systems" *IEEE Parallel & Distributed Technology*, summer 1994, pp. 57- 61

[6] Hwa-Sung Kim, "Genetic & Simulated Annealing based Scheduling in High Speed Network based Distributed Parallel Systems", ICACT' 99, Feb. 1999, pp. 480-485

김 화 성(Hwa-sung Kim) 정회원

1981년 2월 : 고려대학교 전자공학과졸업
1983년 2월 : 고려대학교 전자공학과석사
1996년 10월 : Lehigh univ. 전산학 박사

1984년 3월~2000년 2월 : ETRI 책임 연구원
2000년 3월~현재 : 광운대학교 전자공학부 교수
<주관심 분야> 차세대 인터넷 구조, 미들웨어 환경, 그리드 컴퓨팅